PATENTS • DESIGNS • COPYRIGHT • TRADE MARKS
The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed M Parker

Dated 24 July 2001

CERTIFIED COPY OF PRIORITY DOCUMENT

Patents Form 1/77

nts Act 1977

The Patent Office

1/77

THE PATENT OFFICE
L
18 JUL 2000
NEWPORT

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

18JUL00 E553479-4 000333
P01/7700 0.00-0017481.3

1. Your reference MPS/7548

2. Patent application number *(The Patent Office will fill in this part)* **0017481.3** 18 JUL 2000

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

Bit Arts Limited
Vernon House, 18 Friar Lane,
Nottingham, NG1 6DQ.

Patents ADP number *(if you know it)* 7942410001

If the applicant is a corporate body, give the country/state of its incorporation United Kingdom

4. Title of the invention

Digital Data Protection Arrangement

5. Name of your agent *(if you have one)*

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

Swindell & Pearson

48 Friar Gate,
Derby DE1 1GY

Patents ADP number *(if you know it)* 00001578001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*

a) any applicant named in part 3 is not an inventor, or

b) there is an inventor who is not named as an applicant, or

Yes

Patents Form 1/77

8. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	0
Description	13
Claim(s)	0
Abstract	0
Drawing(s)	5 x 5

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature Swindell & Pearson Date 17/07/2000

Swindell & Pearson

12. Name and daytime telephone number of person to contact in the United Kingdom: Mr. M.P. Skinner - 01332 367051

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

Digital Data Protection Arrangement

The present invention relates to protection arrangements for digital data.

Digital data, including software files and data files, may need protection for various reasons. For instance, protection may be required against unauthorised copying. In the case of software, protection may be required against corruption by virus attack, hacking or the like.

The invention seeks to provide an improved protection arrangement for digital data.

The invention provides a digital data protection arrangement comprising executable code which incorporates sufficient information relating to the protected data to be able to create, when executed, further code which contains the protected data in usable form.

The arrangement may comprise security means operable to detect corruption of the protected data, the further code incorporating instructions to call the security means to assess any corruption. The further code may comprise at least one instruction to jump to execution of the security means, whereby corruption is assessed when that instruction is reached. Preferably the jump instruction is contained within the further code in place of a predetermined set of code, and wherein the security means is operable to recreate the predetermined set of code when executed, to replace the jump instruction. The security means is preferably operable to delete the further code in the event that any corruption is detected.

The arrangement may further comprise relocation means operable to change the location of the security means and to modify the jump instruction to refer to the new location. The relocation means may be contained within the protected data, to operate repeatedly while the protected code is in use.

The executable code may incorporate the protected data in encrypted form, together with executable instructions for decryption. The executable code may contain executable instructions for creating the protected code. The protected data may contain executable code and/or a data file. The arrangement may comprise processing means operable to execute code, and memory means within which the said executable code is stored, the executable code being stored at a memory location indicated within the arrangement as the start point for the protected data, whereby the processor means will cause the said executable code to be executed when seeking to access the protected data, thereby recreating the protected data for subsequent use. The executable code is preferably operable to recreate the protected data in substantially unencrypted form.

The protected data may contain at least one executable instruction which contains a plurality of steps, the steps being executable in more than one order to implement the instruction, and the executable code being operable to create the instruction by creating the steps in an order which changes on each execution of the executable code. The order of the steps is preferably chosen substantially at random on each execution. The steps may include at least one step which initiates the operation of security means operable to detect corruption of the protected data. The executable code may be executed to create the steps on each occasion that the executable instruction is to be executed.

The executable code may be arranged to provide, upon each execution, a part of the protected data in usable form and other data in corrupt form, whereby more than one execution is required to provide the whole of the protected data in usable form. Each part may correspond with a complete executable routine within the protected data, whereby a complete set of routines forming the protected data can be created by repeated execution of the executable code. Each execution of the executable code preferably causes previously created usable code to be corrupted, whereby only the code created by the most recent execution will be in usable form during operation of the

arrangement.

The invention also provides a computer system comprising memory means containing a digital protection arrangement according to any of the preceding definitions.

The invention also provides a data carrier containing software which, when installed on a computer system, is operable as a digital data protection arrangement in accordance with any of the preceding definitions.

The invention also provides computer software which, when installed on a computer system, is operable as a digital data protection arrangement in accordance with any of the preceding definitions.

Examples of the present invention will now be described in more detail, by way of example only, and with reference to the accompanying drawings, in which:-

Fig. 1 illustrates, in a highly simplified form, a computer system for use in implementing the present invention;

Fig. 2 is a highly schematic representation of a known arrangement for storing software in RAM;

Fig. 3 illustrates the use of RAM in accordance with the present invention;

Fig. 4 illustrates the use of RAM in accordance with the present invention and incorporating a relocation arrangement;

Fig. 5 illustrates the use of RAM in accordance with the invention and incorporating a self-modification arrangement; and

Fig. 6 illustrates the use of RAM in accordance with the invention and incorporating arrangements for partial decryption of protected data.

In order to explain the invention, it is useful first to describe briefly a simple computing arrangement with which the present invention may be implemented. Fig. 1 shows a computing arrangement 10 which consists of conventional hardware components, including a processor 12 to which appropriate input and output devices are connected, such as a keyboard 14 and a display 16. Storage is provided in the form of a hard drive 18, or similar bulk memory device, and in the form of RAM (Random Access Memory) 20, for use by the processor 12 during the execution of software.

Fig. 2 illustrates how RAM 20 is conventionally used to run software stored on the drive 18 of an arrangement 10. Initially, the software is loaded by the processor 12 from the drive 18 into RAM 20, by compiling the software on the drive 18 into a sequence of machine code steps 22 (illustrated as STEP 1, STEP 2 etc.), stored in sequence in the RAM 20. The sequence of steps ends with an EXIT instruction 24. It will be appreciated that in a practical example, the number of steps may vary from that shown, and will usually be very much greater in number.

The first step, STEP 1, is conventionally preceded by a LOADER block of code 26 which, when executed, causes other resources to be made available to the software in the RAM, such as drivers for input/output devices 14, 16, or other software resources commonly shared by various different application programs.

In the course of compiling the software to the form illustrated in Fig. 2, the arrangement 10 will record a start point for the block of executable data illustrated, i.e. the memory location in the RAM 20 at which the LOADER 26 begins. The noting of this start point is illustrated schematically in Fig. 2 (and in other Figs) by the use of an arrow 28.

When the arrangement 10 is instructed to run the software now contained within the RAM 20, the processor 12 first determines the start point 28 , goes to that location of the RAM 20 and begins to execute the executable data found at that location. Thus, the processor 12 will first execute the LOADER 26, followed by the STEPS 22, eventually finishing at the EXIT step 24.

It is apparent from Fig. 2 that the software in the RAM 20 is exposed in unencrypted form and is therefore vulnerable to attack by virus software, to unauthorised copying, or to analysis by a malevolent user seeking to attack the software by virus creation or otherwise.

Example 1

Fig. 3 illustrates a basic arrangement according to the invention, by which the protection of the software is improved. Fig. 3A illustrates the state of the RAM 20 when software protected in accordance with the invention is first loaded from the drive 18.

In the condition of Fig. 3A, the RAM 20 contains an initial block of executable code called here an ENGINE 30 and located at the start point 28 noted by the processor 12 for the protected software. Below the ENGINE 30 the RAM 20 is empty at 32.

The ENGINE 30 is executable code, as has been stated. However, the ENGINE 30 is not executable to implement the protected software. Instead, the engine 30 is executable in order to recreate the protected software and write this into the empty region 32 of the RAM 20 when the ENGINE 30 is executed. Thus, in a simple form, the ENGINE 30 may consist of an encrypted version of the protected data, together with executable software for decrypting the software and writing it to the region 32.

It is to be clearly understood that in this document, the term encryption and related terms refer to any technique, including compression, by which data

is converted from its usable form into another form which is not immediately usable. Decryption and related terms are used to refer to the reverse process, including the process of decompression.

It is to be noted in Fig. 3A that the engine 30 is situated at the start point 28. Consequently, when the protected software is called, the processor 12 will first go to the location 28 and thus begin to execute the code which forms the ENGINE 30. Initially therefore, the protected data will be recreated in unencrypted form to fill the space 32, as illustrated in Fig. 3B.

As the processor 12 continues to execute the ENGINE 30, the program pointer will eventually reach the beginning of STEP 1, illustrated at 34 in Fig. 3B, by which time, STEP 1 and the remainder of the protected data will have been recreated in the space 32, so that execution of the protected data will then commence.

It will be realised that the execution of the ENGINE 30 constitutes an additional step in comparison with the conventional arrangement illustrated in Fig. 2. However, it is envisaged that the operation of the ENGINE 30 in machine code will be sufficiently swift to prevent this extra step being noticeable to a user.

Having explained this basic operation as illustrated in Fig. 3, it can be seen that the protected data is less vulnerable to copying or analysis than in the conventional arrangement in Fig. 2. This is because the protected data is not available for attack, copying or analysis until the ENGINE 30 has executed, i.e. until the protected data has been called. Simply loading the protected data from the drive 18 to the RAM 20 is not sufficient to expose the protected data to attack, copying or analysis.

While the arrangements of Figs. 3A and 3B improve on the arrangement of Fig. 12, it is apparent that once the ENGINE 30 has run, the protected data is then exposed at Fig. 3B in the same manner as it was exposed once in RAM in

the form of Fig. 2. In order to further improve the protection, the ENGINE 30 can be modified to recreate data in the space 32 in the form shown in Fig. 3C.

Example 2

In the example of Fig. 3C, the RAM 20 initially contains only the ENGINE 30 and empty RAM 32, as illustrated in Fig. 3A. Execution of the ENGINE 30 again fills the space 32. However, two changes are evident by comparison of Fig. 3D and Fig. 3B. First, STEP 2 has been replaced by a JUMP instruction 36, and a PROTECTION block 38 has been inserted after the EXIT instruction 24. This is achieved in the following manner.

While the ENGINE 30 is running to decrypt the protected data, the ENGINE 30 monitors the resulting decrypted code, looking for certain patterns or sequences of code. The ENGINE 30 may be looking for one sequence or more than one sequence. Naturally, the length of the sequence being sought will affect the likelihood of the sequence being located. In this example, the ENGINE 30 is looking for a sequence which corresponds with STEP 2. When this predetermined set of code is located, in this case STEP 2, the engine 30 does not write the predetermined set of code, but instead, writes a JUMP instruction 36, instructing a jump to the PROTECTION block 38. Consequently, when the engine 30 has run, the protected data is not fully visible, therebeing a JUMP instruction 36 in place of STEP 2.

When the protected data runs, the program will, at some point, reach the location at which STEP 2 would be expected to begin, and will see the JUMP instruction 36. Execution of the PROTECTION block 38 then begins. The block 38 has two functions. First, code for STEP 2 will be created (in a manner similar to the operation of the engine 30) and written over the JUMP instruction 36, so completing the decryption of the protected code. The block 38 will also make security checks, such as by consulting conventional virus protection or the like, to ensure that the virus protection has properly run and that no corruption has been detected. In the event that nothing is amiss, the protection block 38

finishes by instructing a jump back to the location from which the jump to the protection block 38 occurred, which is now the start of code for STEP 2. Operation then continues in the normal way, beginning with STEP 2.

Again, it is envisaged that by operating entirely in machine code, this additional step can be so fast as to be unnoticeable to a user.

It can thus be understood that the protected code does not become fully decrypted until the code is running, thus significantly reducing the risk of virus attack, unauthorised copying or malevolent analysis.

If the PROTECTION block 38 finds something amiss when checking, appropriate retaliation may be implemented. In a simple example, particularly effective against virus attack, the block 38 will delete the version of the protected data contained in the RAM 20 (i.e. the whole of the contents of section 32 of the RAM), thereby deleting the virus or effects of a virus which has contaminated the protected code.

It will be apparent that when operating in machine code and in a realistic situation involving very many lines of code, many JUMP instructions 36 may be created by the engine 30 and at locations which will be difficult for a malevolent watcher to predict or to distinguish from jump instructions conventionally found within a complex block of machine code. It therefore becomes much more difficult for a malevolent watcher to analyse the operation of the protection, or for a virus to disable that protection. This protection can be further enhanced by having the engine 30 watch for a block of code which itself changes, perhaps each time a JUMP instruction is inserted, thus making the insertion of the jump instructions pseudo-random to the outside observer.

Example 3

Figs. 4A and 4B illustrate a protection arrangement with some similarities to the arrangement shown in Figs. 3C, D and E. In particular, a

protection block 40 is used.

Fig. 4A illustrates the initial condition when the protected code has been loaded to RAM 20 from the drive 18. In this example, the protected code is a program which is loaded at 42, the start point of which is noted at 28 by the processor 12. In this example, for illustrative purposes only, the start point is indicated as memory location 10,000.

In addition to the program 42, downloading from the drive 18 also installs the protection block 40 starting, as illustrated in Fig. 4A, at memory location 12,000. The protection block 40 incorporates an ENGINE 44, for reasons to be explained.

The protection block 40 operates in a manner similar to the block 38 of Fig. 3D. That is, the program 42 will jump to the block 40, for corruption to be assessed.

In a conventional arrangement, a protection jump of this nature is vulnerable to attack and disabling by a malevolent watcher or virus which watches for jumps, identifies jumps which regularly go to a memory location which contains code which itself appears to be some form of protection code, and then disables the jump instruction by overwriting.

In this example, this type of attack is hindered as follows. When the first jump instruction to the protection block 40 is encountered, the protection block runs to make the corruption assessment and when that has successfully finished, the engine 44 runs. The ENGINE 44 performs two functions. First, the engine 44 re-writes the jump instructions within the program 42 to refer to a different memory location, such as 13,000 in this example. The engine 44 then recreates the protection block 40 (and itself 44), beginning at memory location 13,000. The protection block 40 has therefore moved, and the jump instructions to memory location 12,000 have been overwritten by instructions to jump to location 13,000. In consequence, the jump instructions can be

continually changing while the program 42 is running, thus greatly increasing the difficulty of a malevolent user or virus recognising and disabling jumps to the protection block 40.

The relocated protection block 40 is illustrated in Fig. 4B.

Example 4

Figs. 5A and 5B illustrate a further example of data protection. In this example, an ENGINE 50 is used to create a block of code in an empty RAM section 52 in a manner similar to that described above in relation to Example 1 and Figs. 3A and 3B. Thus, when the protected code is first loaded from the drive 18 to RAM 20, the position to the left of Fig. 5A pertains, in which the engine 50 is installed in RAM, the remainder of which is empty at 52. In this example, the engine 50 is located at the start point of a sub-routine or other subsidiary operation within a main program. When that sub-routine is called, the ENGINE 50 is executed to create steps to fill the region 52. The position then pertains as shown to the right of Fig. 5A, with four steps, STEP 1, STEP 2, etc. in numerical order within the region 52. When the ENGINE 50 has run, the steps (STEP 1 etc) will then run, to execute the sub-routine.

Fig. 5B illustrates the situation when the sub-routine is next called. Initially, as shown to the left of Fig. 5B, the RAM 20 will be in the state left at the end of the first execution of the sub-routine, as illustrated to the right of Fig. 5A. In particular, the ENGINE 50 is still present, at the start of the block of code. Consequently, when the sub-routine is next called, the ENGINE 50 will again execute. However, the ENGINE 50 is written to create the steps in a different order on each occasion, preferably at random or pseudo-random. This can be achieved without adversely affecting the execution of the sub-routine, because many conventional routines, such as a PRINT statement, may be several thousand lines long when written in machine code, virtually any of which lines can be executed in virtually any order. If there are groups of lines which must be written in a particular order, those lines can be treated as a

block, and written as a single step, so that the order within that block is preserved, but the position of that block changes on each execution of the ENGINE 50.

One or more of the STEPS can be written to implement or call a protection routine against virus attack or corruption. If so, that STEP will appear at a different position each time the sub-routine is run. Thus, a malevolent watcher will see code which is continuously changing and is therefore difficult or impossible to analyse in order to locate the protection, for disabling it. Alternatively, if a malevolent watcher recognises a step as being involved in protection, and disables it, such as by overwriting that step, the malevolent intervention will itself be overwritten on the next occasion the sub routine is called, by operation of the ENGINE 50, thus restoring operation of the protection.

Example 5

When the RAM 20 is in the condition shown in Figs. 3B or to the right of Figs. 5A and 5B, the full protected data is visible in decrypted form. It is envisaged that the forms of protection described above will nevertheless protect the code, but it is also recognised that in some circumstances, the exposure of the complete unencrypted code may represent a point of weakness in the protection, however transitory that exposure might be. A further example illustrates how this difficulty can be overcome. Again, Fig. 6A begins with the RAM 20 in the condition illustrated in Fig. 3A, with an ENGINE 60 at the start point of a sub routine, and the RAM 20 being otherwise empty at 62.

When the sub-routine is called, the processor 12 goes to the start point 28 and begins executing the ENGINE 60. On the first execution of the ENGINE 60, three blocks of code are created within the space 62 (in this example; the number 3 is chosen to illustrate the principles of the invention, not to limit the scope of the invention). These blocks are shown at Fig. 6B. Thus, at Fig. 6B, STEP 1 has been created at 64, followed by two blocks 66 of corrupt or

meaningless data illustrated by asterisks. Alternatively, the blocks 66 could be left empty but it is envisaged that the creation of corrupt or bogus data may cause a malevolent watcher or virus more difficulty by providing a greater amount of code for analysis.

STEP 1 as created by the ENGINE 60 finishes with a RETURN statement 68 which returns execution to the beginning of the ENGINE 60. Thus, having executed STEP 1, the ENGINE 60 executes again. The ENGINE 60 will have set a register to record that a first execution has already taken place. Thus recognising that a second execution is underway, the ENGINE 60 creates STEP 2 at 70, together with corrupt blocks 66 before and after the block 70.

Again, the blocks 66 could be left blank.

STEP 2 can now be executed followed by a return at 72 to the ENGINE 60. A third execution of the ENGINE 60 then creates STEP 3 at 74, and two corrupt blocks 66, allowing STEP 3 to be executed.

Consequently, by repeated operation of the ENGINE 60 in this manner, the complete sequence of STEP 1; STEP 2; STEP 3 can be executed by the processor 12 but at no time is the full sequence visible from RAM 20, thus rendering it difficult or impossible to analyse this sequence or to corrupt it or disable it or any protection it contains. Again, a malevolent watcher will see a code which is continuously changing (or "mutating") and is thus virtually impossible to analyse adequately to attack.

Summary

In each of the examples described above, the code within RAM changes during execution of the executable code. In the simplest example (Example 1) the code changes once, to create a routine. In the other examples, the code changes more often and in several examples, a malevolent watcher will see a code which appears to change continuously.

Very many variations and modifications can be made to the arrangements described above. In particular, it will be readily apparent from a full understanding of the description set out above, that various techniques described there can be used together, in various combinations. For instance, the principle of relocation described in connection with Example 3 can be incorporated into Example 5 so that in addition to the steps of Example 5 being created only temporarily, the location at which they are created can be changed, preferably at random, or pseudo-randomly.

In the examples set out above, the protected data has been executable, but the techniques can readily be adapted to protect a non-executable data file.

Whilst endeavouring in the foregoing specification to draw attention to those features of the invention believed to be of particular importance it should be understood that the Applicant claims protection in respect of any patentable feature or combination of features hereinbefore referred to and/or shown in the drawings whether or not particular emphasis has been placed thereon.

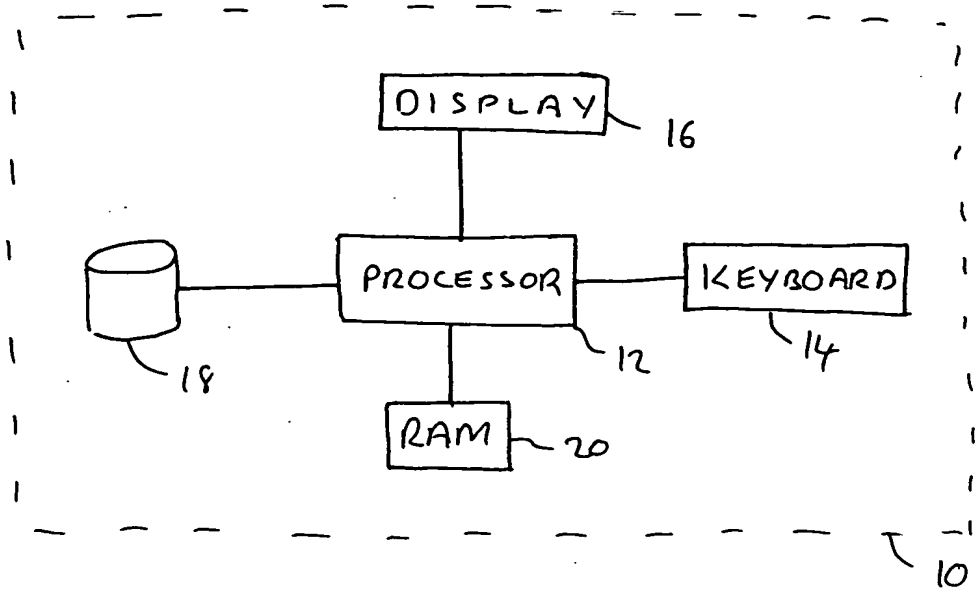
DISPLAY
16
PROCESSOR
KEYBOARD
18
12
14
RAM
20
10

FIG. 1

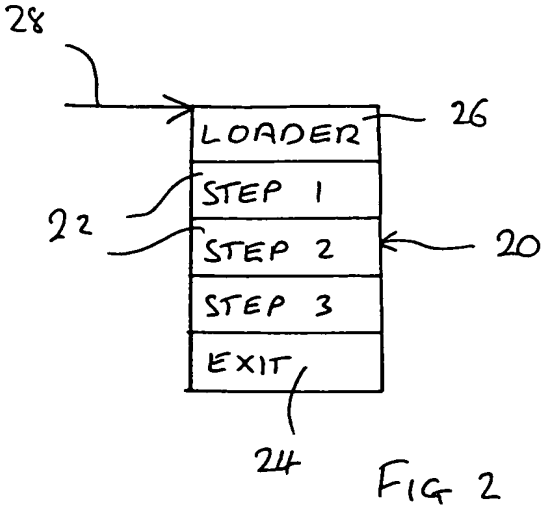
28
LOADER
26
STEP 1
22
STEP 2
20
STEP 3
EXIT
24

FIG 2

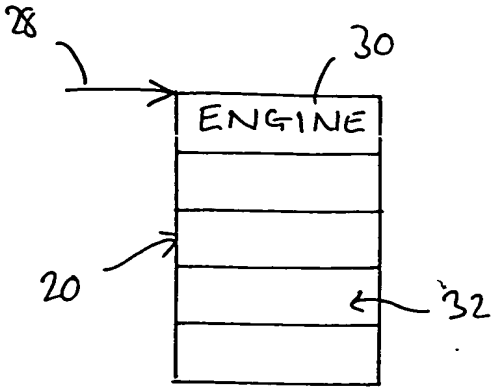

FIG 3a

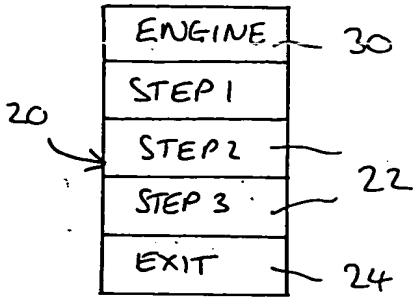

FIG 3b

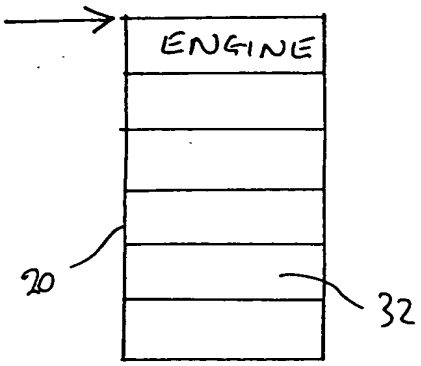

FIG 3c

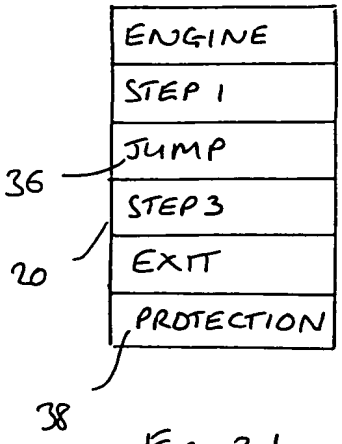

FIG 3d

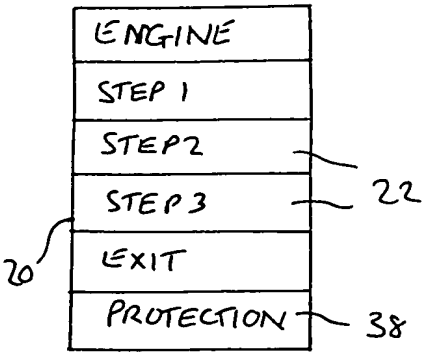

FIG 3e

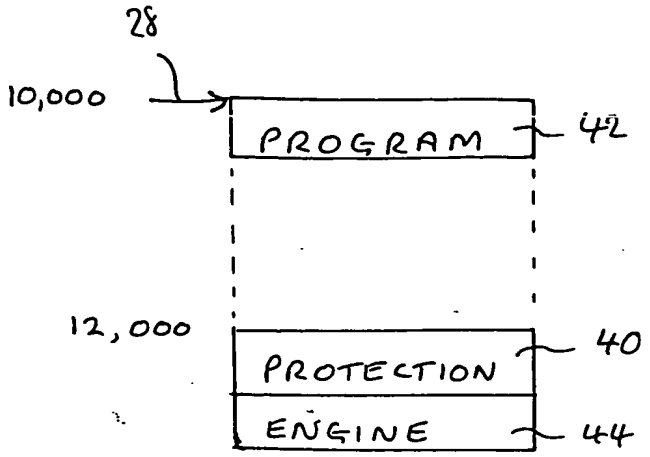
28
10,000
PROGRAM
42
12,000
PROTECTION
40
ENGINE
44

FIG 4A

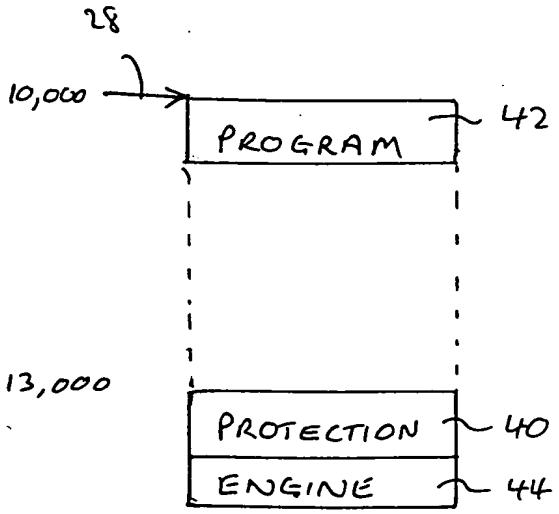
28
10,000
PROGRAM
42
13,000
PROTECTION
40
ENGINE
44

FIG 4B

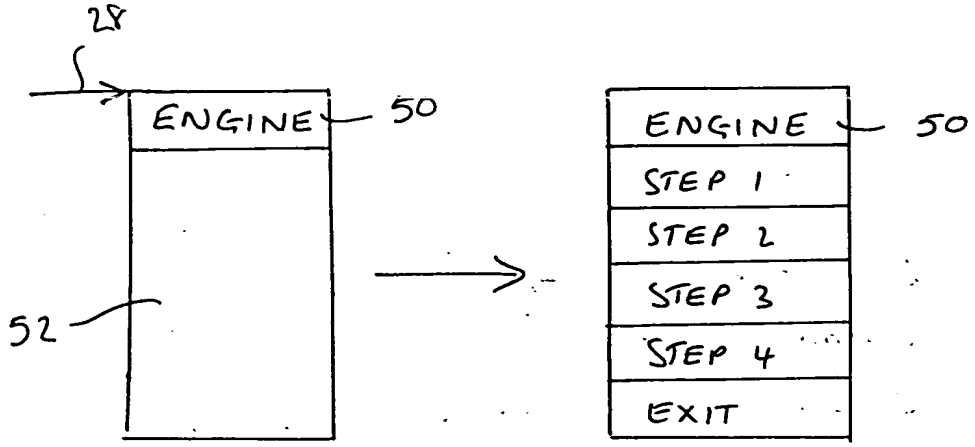
28
ENGINE
50
52
ENGINE
50
STEP 1
STEP 2
STEP 3
STEP 4
EXIT

FIG 5a

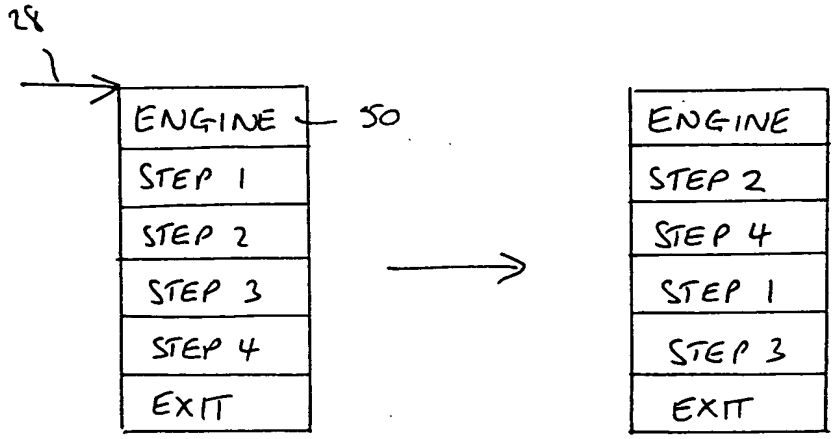
28
ENGINE
50
STEP 1
STEP 2
STEP 3
STEP 4
EXIT
ENGINE
STEP 2
STEP 4
STEP 1
STEP 3
EXIT

FIG 5b

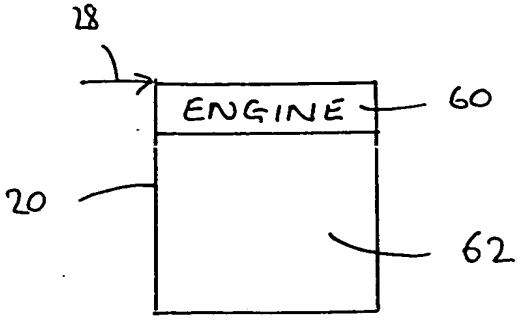

FIG 6a

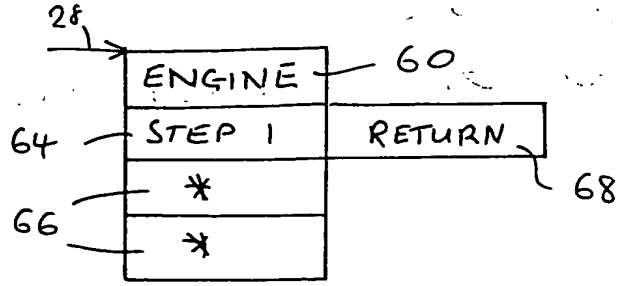

FIG 6b

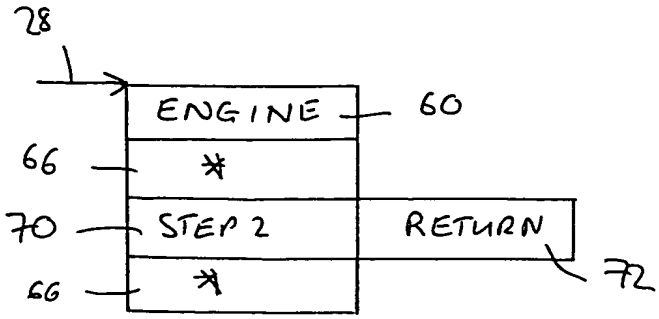

FIG 6c

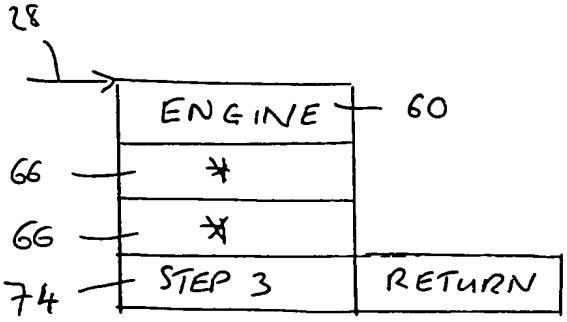

FIG 6d